Java Programming 2081 solution

1. Why do you prefer Java as a programming language? What are the Difference between Procedural and Object-Oriented Programming

Ans -Platform Independent: Java programs run on any device with a JVM (Write Once, Run Anywhere).

Object-Oriented: Promotes modular, reusable, and organized code.

Rich API: Comes with a wide range of libraries and tools.

Robust and Secure: Strong memory management and built-in security features.

Large Community Support: Wide usage and community for help and development.

Difference between Procedural and Object-Oriented Programming

Feature	Procedural Programming (PP)	Object-Oriented Programming (OOP)
Basic Concept	Focuses on a sequence of **procedures** (functions) and the **order** of execution.	Focuses on **objects**, which are instances of **classes** containing both data and functions.
Structure	Code is organized into **functions** or **procedures**.	Code is organized into **classes**, which contain **attributes** (data) and **methods** (functions).
Data Handling	Data is separate from functions and passed between them.	Data is bundled with the methods that operate on it (through **encapsulation**).
Modularity	Modular through **functions**, but limited to function-level reuse.	Highly modular with **objects** and **inheritance**, supporting better **code reusability**.
Reusability	Functions can be reused but with limited scope.	**Inheritance** and **polymorphism** allow classes and objects to be reused and extended.
Design and Maintenance	Linear design; harder to maintain as codebase grows	More flexible and maintainable; code is modular,

Feature	Procedural Programming (PP)	Object-Oriented Programming (OOP)
	due to tight coupling of data and procedures.	making it easier to extend and debug.
Key Principles	**Sequence**, **Selection**, **Iteration** (basic control structures).	**Encapsulation**, **Inheritance**, **Polymorphism**, **Abstraction**.
Examples of Languages	**C**, **Fortran**, **Pascal**, **BASIC**.	**Java**, **Python**, **C++**, **Ruby**, **C#**.

2. What is data type? Explain Different Data Types used in Java

Ans-A data type specifies the type of data a variable can store, such as integer, character, or float.

Data Types in Java

Primitive Types:

int – whole numbers (e.g., 10)

float, double – decimal numbers

char – single character ('A')

boolean – true or false

byte, short, long – integer types with different ranges

Non-Primitive Types:

String – sequence of characters

Array – collection of similar elements

Class, Interface, Object – user-defined types

3. Define array and its types. Write a program in Java Program to find the largest number among 10 inputs using Array.

Ans -An array is a collection of elements of the same data type stored in contiguous memory locations.

Types of Arrays:

1. One-dimensional array – Linear list of elements.

2. Two-dimensional array – Matrix or table.

3. Multidimensional array – More than two dimensions.

```
 java.util.Scanner;
public class LargestNumber {
   public static void main(String[] args) {
      int[] numbers = new int[10];
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter 10 numbers:");
      for(int i = 0; i < 10; i++) {
         numbers[i] = sc.nextInt();
      }
      int max = numbers[0];
      for(int i = 1; i < 10; i++) {
         if(numbers[i] > max) {
            max = numbers[i];
         }
      }

      System.out.println("Largest number is: " + max);
   }
}
```

4. Explain the Features of Object Oriented Programming (OOP). Write down the Difference between Class and Object

Ans-Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to structure software. It provides several key features that make it more modular, reusable, and easier to maintain. The major features of OOP are:

Different beteen Class and Object

Feature	Class	Object
Definition	A **class** is a blueprint or template for creating objects, defining attributes and methods.	An **object** is an instance of a class, containing real data and the ability to use the class's methods.
Nature	A class is a **concept** or **abstract entity**.	An object is a **concrete** instance of a class.
Memory Allocation	A class does not occupy memory on its own, it only defines structure and behavior.	An object occupies memory and holds actual values of attributes defined by the class.
Usage	A class is used to define **properties** and **behaviors** that its objects will have.	An object is used to **store data** and **perform actions** defined by the class.
Example	`class Car { String color; int year; void drive() { ... } }`	`Car myCar = new Car();` — `myCar` is an object of the `Car` class.
Existence	A class exists at the **design** level (or as a template).	An object exists at **runtime** when the class is instantiated.

5. What is method overloading in Java? How to Overload Methods in Java Explain.

Ans- Java's method overloading feature enables you to specify numerous methods with the same name but different parameters in the same class. The quantity and types of parameters supplied influence the method the compiler calls. By offering various methods for carrying out comparable tasks, you can increase the flexibility and readability of your code.

Explain

1. By Changing the number of Parameters

```
class Calculator {
   // Method that adds two integers
   int add(int a, int b) {
      return a + b;
```

```
    }

    // Overloaded method that adds three integers
    int add(int a, int b, int c) {
        return a + b + c;
    }
}

public class Test {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println(calc.add(5, 10));  // Output: 15
        System.out.println(calc.add(5, 10, 15));  // Output: 30
    }
}
```

2. By Changing the Type of Parameters

```
class Test {
    void show(int a) {
        System.out.println("int: " + a);
    }

    void show(String a) {
        System.out.println("String: " + a);
    }
```

```
    public static void main(String[] args) {
        Test t = new Test();
        t.show(10);       // Output: int: 10
        t.show("Hello");   // Output: String: Hello
    }
}
```

3. By Changing the order of Parameters

```
class Test {
    void show(int a, String b) {}
    void show(String b, int a) {}
}
```

6. Define Interface. Write a Program to add two complex numbers (by passing object as an argument)

Ans-An interface is a reference type in Java, similar to a class, that can contain abstract methods. It is used to achieve multiple inheritance and abstraction.

Program

```
class Complex {
    int real, imag;
    Complex(int r, int i) {
        real = r;
        imag = i;
    }
    void add(Complex c) {
        int r = this.real + c.real;
        int i = this.imag + c.imag;
        System.out.println("Sum = " + r + " + " + i + "i");
```

```
    }
    public static void main(String[] args) {
        Complex c1 = new Complex(3, 4);
        Complex c2 = new Complex(1, 2);
        c1.add(c2);
    }
}
```

7. Difference between Exceptions and Errors .Explain try catch, throws and finally

Ans- between Exceptions and Errors

Feature	Exceptions	Errors
Definition	Exceptions represent **unexpected** conditions or issues that can be **handled** by the program.	Errors represent **serious issues** that usually cannot be handled by the program and are typically beyond the program's control.
Type	Used for **recoverable** conditions during runtime.	Used for **irrecoverable** conditions that usually occur due to system-level problems.
Example	`IOException`, `NullPointerException`, `ArithmeticException`	`OutOfMemoryError`, `StackOverflowError`, `VirtualMachineError`
Cause	Caused by programming errors (like bad input, invalid data) or external events (like network failure).	Caused by severe system failures like lack of memory, JVM issues, or hardware failure.
Handling	Exceptions can be **caught and handled** using `try-catch` blocks.	Errors are typically **not caught or handled** because they indicate serious issues that the program cannot fix.
Recoverability	Exceptions are usually **recoverable** if properly handled.	Errors are usually **non-recoverable** and lead to program termination.
Hierarchy	`Throwable → Exception → RuntimeException`	`Throwable → Error`

Feature	Exceptions	Errors
Checked vs Unchecked	Exceptions are either **checked** (require explicit handling) or **unchecked** (runtime exceptions).	Errors are typically **unchecked** (i.e., no need to catch them).
Example of Handling	`try-catch` can be used to handle exceptions.	Errors generally lead to termination and are not typically handled in a `try-catch`.

Example:

```
public class Example {
   static void divide(int a, int b) throws ArithmeticException {
      System.out.println(a / b);
   }
   public static void main(String[] args) {
      try {
         divide(10, 0);
      } catch (ArithmeticException e) {
         System.out.println("Cannot divide by zero");
      } finally {
         System.out.println("Execution completed.");
      }
   }
}
```

8. What are packages? Explain with exmple How to read and write to a file in Java

Ans-A package is a group of related classes and interfaces. It helps in organizing code and preventing name conflicts.

Example:

```
package mypackage;
public class MyClass {
    public void show() {
        System.out.println("Hello from package!");
    }
}
```

```
import java.io.*;
public class FileExample {
    public static void main(String[] args) {
        try {
            // Writing to a file
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Hello, Java File!");
            writer.close();

            // Reading from a file
            BufferedReader reader = new BufferedReader(new
FileReader("output.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
```

}

2080 Regular solution

1. Explain Java as a Programming Platform. List different java Buzzwords

Ans- Java as a Programming Platform: Java is both a programming language and a platform. As a platform, Java provides a runtime environment (JRE) and a set of APIs to develop and run applications across different operating systems. Java applications are compiled into bytecode which runs on the Java Virtual Machine (JVM), making them platform-independent.

different java Buzzwords

1. **Simple** – Easy to learn and use.
2. **Object-Oriented** – Based on OOP principles.
3. **Platform Independent** – Write once, run anywhere.
4. **Secure** – Provides a secure runtime environment.
5. **Robust** – Strong memory management and exception handling.
6. **Multithreaded** – Supports multiple threads of execution

1. b). What are Operators in Java ?Explain Arithmetic and comparision Operators with suitable example.

Ans-In Java, **operators** are **special symbols** used to **perform operations** on variables and values. They are essential in expressions and help manipulate data and control the flow of programs.

✓ 1. Arithmetic Operators in Java

Arithmetic operators are used to perform **basic mathematical operations** such as addition, subtraction, multiplication, division, and modulus.

Operator	Description	Example (if a = 10, b = 5)	Result
+	Addition	`a + b`	15
−	Subtraction	`a - b`	5

Operator	Description	Example (if a = 10, b = 5)	Result
*	Multiplication	a * b	50
/	Division	a / b	2
%	Modulus (remainder)	a % b	0

◆ **Example Code:**

```
public class ArithmeticExample {
    public static void main(String[] args) {
        int a = 10, b = 5;

        System.out.println("Addition: " + (a + b));
// 15
        System.out.println("Subtraction: " + (a -
b));      // 5
        System.out.println("Multiplication: " + (a *
b));   // 50
        System.out.println("Division: " + (a / b));
// 2
        System.out.println("Modulus: " + (a % b));
// 0
    }
}
```

2**. a) a. Define class and object. Write a program to define a class box with data member 1, b and h. Define member function to read its data member and calculate volume of box.**

Ans-Class :- A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

➤**Object:-** Objects have states and behaviors. Example: A dog has states color, name, breed as well as behaviors - wagging the tail, barking, eating. An object is an instance of a class.

Program:

```
import java.util.Scanner;

class Box {
```

```
    // Data members
    int l, b, h;
    // Method to read data
    void readData() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter length: ");
        l = sc.nextInt();
        System.out.print("Enter breadth: ");
        b = sc.nextInt();
        System.out.print("Enter height: ");
        h = sc.nextInt();
    }
    // Method to calculate and display volume
    void calculateVolume() {
        int volume = l * b * h;
        System.out.println("Volume of the box = " + volume);
    }
}
public class Main {
    public static void main(String[] args) {
        Box box1 = new Box();       // Create object
        box1.readData();           // Read box dimensions
        box1.calculateVolume();    // Calculate and display volume
    }
}
```

2. b)Define Loop. write a program to print n Prime Number.

Ans- A **loop** is a control structure in programming that **repeats a block of code** as long as a specified condition is true. Loops help reduce code repetition and automate repetitive tasks.

Program to Print n Prime Numbers:

```
import java.util.Scanner;
public class PrimeNumbers {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter n: ");
      int n = sc.nextInt();
      int count = 0, num = 2;
      while (count < n) {
         boolean isPrime = true;
         for (int i = 2; i <= num/2; i++) {
            if (num % i == 0) {
               isPrime = false;
               break;
            }
         }
         if (isPrime) {
            System.out.print(num + " ");
            count++;
         }
         num++;
      }
   }
}
```

3. a) Define exception. Explain Exception Handling Mechanism with example

Ans - An **exception** in Java is an **unexpected event** or **error** that occurs during the **execution of a program**, disrupting the normal flow of instructions.

✓ Exception Handling Mechanism in Java

Exception Handling in Java is a powerful mechanism that allows a program to **detect**, **handle**, and **recover** from runtime errors (called exceptions), without terminating the program unexpectedly.

◆ Why Exception Handling?

- To maintain **normal flow** of the program during runtime errors.
    - To catch and manage **unpredictable events** (like divide by zero, file not found, etc.).

✓ Example:

```
public class ExceptionExample {
   public static void main(String[] args) {
      try {
         int a = 10, b = 0;
         int result = a / b;  // This will cause ArithmeticException
         System.out.println("Result = " + result);
      } catch (ArithmeticException e) {
         System.out.println("Error: Cannot divide by zero.");
      } finally {
         System.out.println("Program ended.");
      }
   }
}
```

3.b)What is abstract Method?write a program to implement abstract method .

Ans- An **abstract method** in Java is a method that is **declared without a body** (no implementation). It is used when the method should be **implemented by subclasses**.

Program:

```
abstract class Animal {
   abstract void sound();
}
class Dog extends Animal {
   void sound() {
      System.out.println("Dog barks");
   }
   public static void main(String[] args) {
      Animal obj = new Dog();
      obj.sound();
   }
}
```

4.a) What are threads?Explain thread States

Ans-A **thread** is a **lightweight sub-process** or **smallest unit of a process** that can run **independently and concurrently** with other threads. Java allows **multithreading**, which means running **multiple threads simultaneously**, improving performance, especially in multitasking environments.

Explain thread States

1. New

- A thread is in the **New** state when it is **created** but not yet started.
- Created using:
- `Thread t = new Thread();`

2. Runnable

- After calling `start()`, the thread enters the **Runnable** state.
- It is **ready to run** but waiting for CPU time.

```
t.start();  // Moves thread from New to Runnable
```

🔄 3. Running

- The thread is in the **Running** state when the **CPU starts executing** its `run()` method.
- Only one thread runs at a time per processor (in reality, time-sliced).

⛔ 4. Blocked

- A thread enters the **Blocked** state when it tries to access a **locked resource** used by another thread.
- It waits until the resource becomes available.

⍰ 5. Waiting

- A thread is in **Waiting** state if it waits **indefinitely** for another thread to perform a task.
- Example: `wait()`, `join()` without timeout.

⏳ 6. Timed Waiting

- Thread waits for a **fixed period of time**.
- Methods: `sleep(time)`, `join(time)`, `wait(time)`.

❌ 7. Terminated (Dead)

- A thread is in the **Terminated** state when it:
  - **Completes execution**, or
  - Is **forcibly stopped** due to an error.

4.b) b. What are java applets? Write its benefits. Explain embedding applet to HTML file with example.

Ans- A **Java Applet** is a **small Java program** that is embedded into a **web page** and runs in a **web browser**. It was mainly used to create **interactive features** like animations, games, charts, and forms in websites.

Embedding Applet in HTML:

<applet code="MyApplet.class" width="300" height="300"></applet>

Java Code:

```
import java.applet.Applet;
import java.awt.Graphics;
public class MyApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello Applet", 50, 50);
   }
}
```

5. a) What are I/O streams? Explain how we can read and write format to a file in java .

Ans - I/O Streams in Java refer to the mechanism used for **input and output operations**, such as **reading from** or **writing to** files, memory, keyboard, or network.

- I/O stands for **Input/Output**.

Java uses **streams** to handle all I/O operations in a **standardized** and **efficient** way.

📝 1. Writing Formatted Data to a File

```
import java.io.PrintWriter;
import java.io.IOException;
public class WriteToFile {
   public static void main(String[] args) {
      try {
         PrintWriter writer = new PrintWriter("data.txt");
         String name = "John";
         int age = 25;
         double salary = 45678.90;
         // Writing formatted data
```

```
            writer.printf("Name: %s\nAge: %d\nSalary: %.2f", name, age, salary);


            writer.close();  // Close file after writing
            System.out.println("Data written successfully.");
        } catch (IOException e) {
            System.out.println("Error writing to file.");
        }
    }
}
```

5.b) Define interfaces. Write a program to implement interfaces.

Ans- An **interface** in Java is a **reference type** that is similar to a class but only contains **abstract methods** (method declarations without bodies) and **constants** (by default `public`, `static`, and `final`).

✓ Program to Implement Interface in Java

```
// Define an interface
interface Animal {
    void makeSound();  // abstract method
    void eat();
}
// Implement the interface in a class
class Dog implements Animal {
    // Provide implementation for interface methods
    public void makeSound() {
        System.out.println("Dog barks");
```

```
    }
    public void eat() {
        System.out.println("Dog eats bones");
    }
}
// Main class to run the program
public class InterfaceExample {
    public static void main(String[] args) {
        Dog d = new Dog();  // create object
        d.makeSound();
        d.eat();
    }
}
```

6.) a. Define method overriding. Write any program to implement concept of method overriding

Ans-Method overriding is a feature in object-oriented programming where a subclass provides a specific implementation of a method that is already defined in its superclass.

When a method in a subclass has the same name, return type, and parameters as a method in the parent class, the subclass's method **overrides** the parent class's method. This allows the subclass to customize or replace the behavior of that method.

Program:

```
class Animal {
    void sound() {
```

```
        System.out.println("Animal sound");
    }
}
class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound(); // Outputs "Dog barks"
    }
}
```